An Integrated Environment for Efficient Formal Design and Verification

Final Report

IN-61

[illegible]

Administrative information

ARPA order number:	A721
Grant number:	NAG 2-891
Actual Performance period:	January 1, 1994 – December 31, 1998
Agent:	NASA
Contract title:	"An Integrated Environment for Efficient Formal Design and Verification"
Organization:	Stanford University
Subcontractors:	SRI International

Introduction

The general goal of this project was to improve the practicality of formal methods by combining techniques from model checking and theorem proving. At the time the project was proposed, the model checking and theorem proving communities were applying different tools to similar problems, but there was not much cross-fertilization. This project involved a group from SRI that had substantial experience in the development and application of theorem-proving technology, and a group at Stanford that specialized in model checking techniques.

Now, over five years after the proposal was submitted, there are many research groups working on combining theorem-proving and model checking techniques, and much more communication between the model checking and theorem proving research communities. This project contributed significantly to this research trend.

The research work under this project covered a variety of topics: new theory and algorithms; prototype tools; verification methodology; and applications to problems in particular domains.

New theory and algorithms

Research in model checking has emphasized highly automatic verification methods. Theorem provers have partially automated the proof process, but still rely heavily on user interaction to complete proofs. The need for more automation in theorem proving and more generality in model checking became clear early in the project.

In response to this need, a decision procedure for a theory of bit-vectors encompassing bit-field extraction, bit-vector concatenation, and equality was invented at SRI. Based on this, a more procedure using a somewhat different approach was then developed at Stanford, which dealt with arithmetic on bitvectors as well as concatenation and extraction. These developments were inspired

Rec'd JUN 18 1999

CC: 202A-3 ✓

CASI

by the need to reason about bitvectors in the verification of microprocessor designs (including AAMP-FV and the FLASH protocol processor, discussed below).

There were also other improvements to decision procedures, including an improved understanding of several previous approaches to cooperating decision procedures, and a more general treatment of arrays. The Stanford group did the initial development of an efficient decision procedure, called SVC, for quantifier-free first-order logic formulas, with linear arithmetic, uninterpreted functions, arrays, and bitvectors, for general use, while the SRI group revised the decision procedures in PVS.

In response to the need to generalize model checking, an automatic abstraction mechanism for systems with an variable number of similar processes. The abstraction method keeps track of whether zero, one, or several processes is in a particular state (e.g., at a particular location in their programs), abstracting away from the exact number of processes in a particular state, as well as the order of the processes. This abstraction retains enough information to verify important properties of several types of protocols, while reducing potentially infinite state spaces to finite spaces.

Verification Methodology

It became clear during the project that a lot of research needed to be done on how to use verification tools, not just on the development of the tools themselves. The simplest idea for combining model checking and theorem proving was quite successful: debug the description and specification with a model checker before verifying the general case with a theorem prover. Theorem provers are better at proving the correctness of correct systems than at finding bugs in incorrect systems, while model checkers are good at finding bugs and providing useful debugging information, but not good at proving correctness in the general case (which often requires reasoning about infinite numbers of states). This approach was applied successfully to several cache coherence protocols and distributed algorithms. To make this approach simpler, we developed a translator from our Murphi language to a representation of the same behavior in PVS.

Another, more integrated, approach was to use PVS to prove the soundness of abstractions that reduced an infinite state problem to a finite state problem. A mu-calculus model checker was embedded in PVS, which was used to check the abstracted protocol. This approach was used quite successfully to verify the Phillips bounded retransmission protocol and inspired much subsequent work, both in this project and at other institutions, to further automate the construction of abstractions.

One very general and influential idea was a method for defining abstraction functions. We first observed in microprocessor verification that the main difficulty in defining an abstraction function was dealing with partially completed operations; for example, a simple pipelined processor might have 4 or 5 instructions at any given time that have been started but not completed. We found that it was often possible to define a *completion* function relatively, which simulated the effect of finishing the uncompleted actions, which was the hard part of defining an abstraction function.

For processor verification, the completion function could sometimes be defined semi-automatically, by symbolically executing the processor description in a mode that completes pending instructions without starting new ones. In other cases, it was simple to define the function manually. We also found that the same idea could be applied more generally to certain types of protocols, when the protocols were intended to simulate atomic transactions while implementing them as a series of smaller steps. For protocols, the idea was also to define a *aggregation* function that completed all pending transactions (but, unlike a pipelined microprocessor, protocol transactions are not necessarily ordered). Using the aggregation approach, we were able to verify the FLASH cache coherence protocol using PVS, something that was not previously possible.

The aggregation approach could also be used with model checking. The user defines an aggregation function, and checks that it has the desired properties automatically using the model checker. This capability provided a way to debug a protocol and aggregation function using a model checker, then use the same aggregation function to verify the protocol for an unbounded number of processes using a theorem prover.

Work at SRI and the University of Utah led to development of a method that shares some of the motivation and insight of this approach, but differs in its technology and application. This method was applied successfully to several advanced pipeline architectures (with out of order execution and reorder buffers), yielding much simpler and more automated proofs than previous approaches.

In the latter stages of the project, we investigated several approaches to *predicate abstraction*. Predicate abstraction was originally proposed by French researchers (who used the decision procedures in our PVS prover). We prototyped several improved schemes for predicate abstraction using PVS, and also built a more efficient prototype tool using SVC and Boolean decision diagrams (or BDDs, a widely-used data structure for boolean functions). This preliminary work was the basis for the DARPA-funded SAL project, which is currently underway.

Finally, we received some additional funding from NASA in the last few months to investigate the possibility of model checking concurrent software. We adapted a Java translator from NASA to generate input for our Murphi model checker, and then did a case study of a multi-threaded game server written in Java. After a series of manual abstractions, we were able to discover a deadlock in the game server using the Spin model checker as a back-end verifier. One conclusion of this work was that Murphi had some shortcomings as a Java verification tool, particularly its limitation to static data structures and a fixed number of threads.

Tool Development

The efforts to integrate our existing methods, and apply them to challenging problems, inspired improvements to old tools and the development of new tools. In addition to the enhancements to Murphi mentioned above, we implemented a parallel version of Murphi, which exhibited almost linear speedup even on a system with 64 processors, and a probabilistic verification method that saved memory by storing small signatures of states instead of full states in a table. PVS was enhanced in many ways, including speeding up the rewriting (which turned out to be a bottleneck in processor verification efforts), and various improvements to the internal decision procedures.

The most effective enhancements to PVS were the incorporation of algorithms and datastructures from the model checking world. An efficient BDD package was embedded in PVS, resulting in multiple orders-of-magnitude speedups on problems involving significant combinational logic. The embedded mu-calculus model checker mentioned earlier was based on the embedded BDDs. PVS is freely distributed and is now in use at hundreds of industrial and academic sites worldwide.

We also began implementation of a more efficient library for deciding formulas in quantifier-free first-order logic, called SVC (eventually, development of SVC was continued under another contract). SVC was written in C++, and was based on the method of integrating cooperating decision procedures devised by Shostak (whose procedures were used in PVS). SVC emphasizes efficiency in its algorithms and data structures. One important optimization was using a DAG representation for expressions, which shares common sub-expressions. SVC was used for much of our processor verification work, and was also used to construct a tool for analyzing RSML specifications of safety critical systems (the RSML work was funded under another contract). SVC is now being freely distributed in source form, and is being used at several other sites in the U.S. and Europe.

Applications

As planned, a significant amount of the effort in the project went into studying specific examples. Experience with examples inspired new directions in decision procedures, verification tools, and verification methodology.

Many of the examples were cache coherence problems of various types. We started working on these problems in response to comments from the DASH multiprocessor project at Stanford. Over time, Murphi was modified incrementally to make verification of cache coherence protocols more convenient. Our work also got the attention of other people who were working on cache coherence. Specifically, "counting abstraction" was incorporated into Murphi because similar techniques had been shown to be effective on cache coherence protocols.

We also were able to use our knowledge of cache coherence to drive research in other verification techniques. The aggregation methodology mentioned above was developed as part of our effort to verify the FLASH cache coherence protocol using PVS.

Over the course of the project, our ability to verify cache coherence protocols went from small protocols with three processors, one address, and one bit of data, to unbounded numbers of processors, addresses and data. Furthermore, Murphi was used by several other groups for verifying cache coherence protocols. Murphi was incorporated into the Tempest/Teapot system at the University of Wisconsin for verifying cache coherence protocols that could be customized by users (who could also introduce bugs). It not only found bugs in custom protocols, it found a bug in one of the pre-defined protocols. We were asked to develop an executable specification using Murphi of the RMO memory model for the widely-used SPARC V.9 architecture; we ended up participating in the design, and using Murphi to verify most of the illustrative examples that were published in the architecture definition. Murphi was also used at IBM Research in the development of the cache coherence protocols in ASCI blue, one of the fastest computers currently being designed.

Murphi was also useful for other verification applications. One of the most interesting (done in conjunction with a security project at Stanford) was automated analysis of cryptographic protocols (e.g. authentication protocols). The protocol rules are modeled in Murphi, along with a model of a hostile intruder that can see the contents of messages, kill messages, and insert new messages into the communications medium. The actual encryption is modeled abstractly (the intruder can learn the contents of a message if it has seen the decryption key in the clear). The "secure sockets layer" (SSL) protocol, which is widely used in internet commerce, was modeled and verified using this approach. All known problems in the protocol were discovered, along with several previously unpublished anomalies.

Microprocessor design verification was another area where this project was quite successful. Both research groups worked on this problem, using several different approaches. At SRI, work on the AAMP-5 and AAMP-FV processors stimulated several performance improvements in PVS, as well as the development of an extensive theory of bit-vectors. Researchers at Rockwell-Collins used the technology developed in the AAMP work as the basis for a novel approach to microcode validation that was employed in development of their JEM1 processor (the world's first direct-execution Java processor).

A formal derivation and proof of the SRT division algorithm was also developed using PVS (a bug in the SRT division algorithm was responsible for the infamous Pentium division bug). This work was further developed by researchers at NASA to give a general treatment to a wide class of subtractive division algorithms.

The first effort to verify a simple microprocessor design by the Stanford group yielded results which have been widely influential. The basic technique involved symbolically simulating a two high-level descriptions of the design, and then comparing the results using SVC. Several important and fundamental ideas came out of this project: the use of uninterpreted functions to abstract away the details of a data path, defining abstraction functions by symbolically flushing out temporary state, and the use of decision procedures for quantifier-free first order logic. Other researchers have used these ideas directly or adapted them to other techniques (such as BDD based symbolic simulation). A commercial microprocessor for digital signal processing at Motorola was formally verified by another group using this basic approach.

SVC has proven to be useful for check specifications of safety-critical systems on another project. Specification languages for safety critical systems often insist that the specifications be *deterministic*, meaning that only one transition can be enabled at a time. However, the enabling conditions for transitions can be complex, so checking determinism can be difficult. We prototyped a tool based on SVC to check for determinism of selected transitions in RSML specifications, and applied to tool to parts of an existing RSML specification for the TCAS collision avoidance system (which is required in all commercial aircraft carrying over 30 passengers). Using the tool, we discovered a serious problem in the specification (it turned out that the problem had been discovered independently by a validation group, and fixed in a later revision of the specification).

References

[1] Natarajan Shankar. Unifying verification paradigms. In Bengt Jonsson and Joachim Parrow, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1135 of *Lecture Notes in Computer Science*, pages 22–39, Uppsala, Sweden, September 1996. Springer-Verlag.

[2] John Rushby. Automated deduction and formal methods. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, number 1102 in Lecture Notes in Computer Science, pages 169–183, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[3] Harald Rueß. Hierarchical verification of two-dimensional high-speed multiplication in PVS: A case study. In Mandayam Srivas and Albert Camilleri, editors, *Formal Methods in Computer-Aided Design (FMCAD '96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 79–93, Palo Alto, CA, November 1996. Springer-Verlag.

[4] H. Rueß, N. Shankar, and M. K. Srivas. Modular verification of SRT division. *Formal Methods in Systems Design*, 14(1):45–73, January 1999.

[5] David Cyrluk, Patrick Lincoln, and N. Shankar. On Shostak's decision procedure for combinations of theories. In M. A. McRobbie and J. K. Slaney, editors, *Automated Deduction—CADE-13*, number 1104 in Lecture Notes in Artificial Intelligence, pages 463–477, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[6] Ravi Hosabettu, Mandayam Srivas, and Ganesh Gopalakrishnan. Decomposing the proof of correctness of pipelined microprocessors. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer-Aided Verification, CAV '98*, volume 1427 of *Lecture Notes in Computer Science*, pages 122–134, Vancouver, Canada, June 1998. Springer-Verlag.

[7] Ulrich Stern and David L. Dill. Using magnetic disk instead of main memory in the Murφ verifier. In *10th International Conference on Computer Aided Verification*, June 1998.

[8] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using murphi. In *IEEE Symposium on Security and Privacy, May 1997*, 1997.

[9] Seungjoon Park and David L. Dill. Verification of cache coherence protocols by aggregation of distributed transactions. *Theory of Computing Systems*, 31(4):355–376, 1998.

[10] Seungjoon Park, Satyaki Das, and David L. Dill. Automatic checking of aggregation abstractions through state enumeration. In *IFIP TC6/WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification*, November 1997.

[11] Seungjoon Park and David L. Dill. Protocol verification by aggregation of distributed transactions. In *Conference on Computer-Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 300–310. Springer-Verlag, July 1996.

[12] Seungjoon Park and David L. Dill. Verification of flash cache coherence protocol by aggregation of distributed actions. In *Symposium on Parallel Algorithms and Architectures*, pages 288–296, June 1996.

[13] Seungjoon Park and David L. Dill. An executable specification, analyzer and verifier for RMO (relaxed memory order). In *IEEE Transactions on Computers*, 1999. Accepted for publication.

[14] Seungjoon Park and David L. Dill. An executable specification, analyzer and verifier for RMO (relaxed memory order). In *Symposium on Parallel Algorithms and Architectures*, 1995.

[15] C. Norris Ip and David L. Dill. Verifying systems with replicated components in Murφ. *Formal Methods in System Design*, 1998. Accepted for publication.

[16] Norris C-W. Ip and David L. Dill. Verifying systems with replicated components in murϕ. In *Conference on Computer-Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 147–158. Springer-Verlag, July 1996.

[17] Clark Barrett, David Dill, and Jeremy Levitt. Validity checking for combinations of theories with equality. In Mandayam Srivas and Albert Camilleri, editors, *Formal Methods In Computer-Aided Design*, volume 1166 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, November 1996. Palo Alto, California, November 6–8.

[18] Jerry R. Burch and David L. Dill. Automatic verification of pipelined microprocessor control. In David L. Dill, editor, *Conference on Computer-Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 68–80. Springer-Verlag, 1994. Stanford, California, June 21–23, 1994.